Albrecht Weinert

Tutorial

AJAX mit GWT — Tipps und Tricks

Stand: 15. Juni 2007

Albrecht Weinert
Labor für Medien und verteilte Anwendungen (MEVA-Lab)
Fachbereich Informatik der FH Bochum

AJAX mit GWT — Tipps und Tricks

V1.00 (20.01.2007 17:31)	:	neu, unvollst.
V1.02 (02.02.2007 15:08)	:	Erg. zu Pfaden/Namen
V1.03 (15.02.2007 14:51)	:	Beta 1
V1.06 (26.08.2007 16:57)	:	kl. Korr.
V1.07 (06.09.2007 18:16)	:	GWT Version von 1.2.22 nach 1.4.60
V1.08 (18.09.2007 10:02)	:	UTF-8 Problem des GWT-Compilers
V1.09 (01.02.2008 15:00)	:	installed extension / jar-Version verdeutl.
V1.10 (06.06.2008 14:31)	:	Hinweise auf GWT1.5.0rc.

Version: V1.10

Zuletzt geändert von A. Weinert am 6. Juni 2008

Inhalt

1. Umfeld und Einordnung

Es geht um Browser, WWW-Seiten, WWW-Server und Webdienste.

Vorausgesetzt wird die Beherrschung und jeweils geeignete Installation von Java und Tomcat. Eclipse und cvsNT gehören werkzeugseitig eigentlich dazu, sind aber für das Folgende nicht unabdingbar. Siehe dazu [3]..[6] im selben Verzeichnis wie dieses Dokument.

AJAX und der teilweise synonym dafür verwendete Begriff Web2 sind in aller Munde.

AJAX heißt „Asynchronous JavaScript + XML“, obgleich JavaScript heute eigentlich ECMA-Script heißt. AJAX funktioniert grob gesagt so:

- I.A. von Nutzereingriffen auf die angezeigte Web-Seite getriggert, senden JavaScript-Funktionen XML-Anfragen an den Web-Server. Diese Anfrage kann beliebige Aktionen im Web-Server, im Allgemeinen über Servlets in einem Tomcat-J2EE-Container, auslösen und so einen Web-Service darstellen.
- Die auch via XML rücktransportierte Antwort wird von einer (anderen) JavaScript-Funktion verarbeitet. I.A. wird diese über das Document-Object-Model (DOM) der HTML-Seite, dazu verwendet, diese partiell zu ändern.

Aus Sicht des Nutzers sind diese clientseitigen kleinen, aber wichtigen Manipulationen der Seite der entscheidende Punkt bei AJAX. Ohne diese clientseitige Manipulation werden alle Änderungen des Inhalts vom Server durch Liefern einer neuen Seite dargestellt.

Aus reiner Benutzersicht wäre es ja egal, wie dies gemacht wird. Allerdings bedingt das Neuladen und Darstellen einer ganzen HTML-Seite, bloß um z.B. zwei Zahlen in einer Tabelle zu aktualisieren, meist eine deutliche Störung. Mit clientseitigen Änderungen durch konsequentes AJAX kann der Anwender hingegen schon den Eindruck gewinnen, an einer lokalen Applikation zu arbeiten und nicht eine Folge von Web-Seiten zu empfangen. Dieser Eindruck, an einer lokalen Applikation zu arbeiten, ist durch die Möglichkeit lokaler Verarbeitung mit JavaScript, ja auch in unterschiedlichem Maße richtig.

Gegenüber dem Ansatz mit Liefern von HTML oder zu transformierenden XML durch den Web-Server oder durch den WWW-Dienst (Servlet) bringt AJAX dem Entwickler

- geänderte und zusätzliche Server-Programmierung,
- Handhabung zusätzlicher Protokolle,
- clientseitige Programmierung mit einer zusätzlichen Programmiersprache, nämlich ECMA-Script bzw. JavaScript.

Und n.b.: JavaScript ist nicht Java!

Und zusätzlich zur Darstellung der eigentlichen Funktion erfordern die Aspekte

- Bedienung unterschiedlicher Browser, insbesondere des
- immer abweichenden Microsoft-Internet-Explorers, und die
- Bookmark- und Back- / Forwardmöglichkeiten

zusätzliche und teilweise komplexe Maßnahmen.

Die Browser-Unabhängigkeit war auch ohne AJAX nicht immer einfach aber doch routinemäßig beherrscht oder mit Inkaufnahme von Schönheitsfehlern beim nicht bevorzugten Browser toleriert. Mit AJAX wird dies erheblich schwieriger.

Bookmark- und History-Funktionen waren praktisch meist gar kein Thema; die Browser lieferten dies zufriedenstellend. AJAX kann bedeuten, dass "stundenlang" dieselbe Seite oft bis zur Nichtmehrwiedererkennbarkeit manipuliert wird. Dann sind diese Features ohne erhebliche Zusatzprogrammierung aus Anwendersicht einfach weg.

Nun gibt es viele käufliche und freie Software, Bibliotheken und Werkzeuge für AJAX.
Das „Google Web Toolkit" (GWT) ist also eine von vielen.
GWT ist frei erhältlich aber nicht open source.

Im Spektrum der AJAX-Designstrategien (nach [1])

Strategy 1: Do It Yourself
Strategy 2: Use a Client-Side JavaScript Technology Library
Strategy 3: Use a Client-Server Framework
Strategy 4: Do the Wrap Thing
Strategy 5: Go Remote
Strategy 6: Go All Java Technology

gehört GWT in die Kategorie 6.

Hier wird man von AJAX-Einzelheiten abgeschirmt, und insbesondere bleibt man von der Programmierung von JavaScript verschont. Hinzu kommt, dass GWT alle oben als erschwerend genannten Aspekte (Browser-Unterschiede, History) ohne weiteres gekonnt handhabt bzw. unterstützt.

2. Installation auf dem Entwicklungsrechner

Von Google (http://code.google.com/webtoolkit/) lädt man sich eine Datei

```
06.09.2007    15:29    19.542.413    gwt-windows-1.4.60.zip
```

oder eine neuere Version und packt sie mit (beispielsweise)

```
cd /d E:\Tests\GWT
jar xfv <woImmerEsSteht>\ gwt-windows-1.x.yz.zip
```

aus. Im aktuellen Pfad entsteht ein Unterverzeichnis gwt-windows-1.4.60. bzw. entsprechend der neueren Version benannt Dieses enthält die Software, die Dokumentation und einige Beispiele.

Hinweis auf neuere Version; Inzwischen (M 2008) gibt es GWT 1.5.0 als

```
06.06.2008    10:07    21.451.374    gwt-windows-1.5.0.zip
```

Die im folgenden beschriebene Installation funktioniert und Alles andere funktioniert damit sinngemäß auch.
Allerdings werden laufende und erprobte Web-Anwendungen bei der Portierung (und Neuübersetzung) von 1.4.60 nach 1.5.0 gebrochen — bis hin zu grauen Feldern anstelle von Knöpfen geht teilweise nichts mehr. Der Übergang kann also aufwändig werden und muss überlegt sein
.

Für die GWT-Software und Dokumentation richtet man sich am besten an passender Stelle ein eigenes Programmverzeichnis ein ...

```
    C:
...md C:\Programme\GWT
```

... und verschiebt dorthin die GWT-Software sowie die GWT-Dokumentation:

```
19.01.2007    17:01         1.169    about.txt
19.01.2007    17:01         3.160    about.html
19.01.2007    17:01         5.961    index.html
19.01.2007    17:01        27.732    release_notes.html
19.01.2007    17:01       276.629    gwt-servlet.jar
19.01.2007    17:01     9.297.529    gwt-dev-windows.jar
19.01.2007    17:01     1.146.806    gwt-user.jar
19.01.2007    17:01        12.800    gwt-ll.dll
19.01.2007    17:01       323.584    swt-win32-3235.dll
19.01.2007    17:01           101    junitCreator.cmd
19.01.2007    17:01           102    projectCreator.cmd
19.01.2007    17:01           106    applicationCreator.cmd
19.01.2007    17:01            99    i18nCreator.cmd
22.01.2007   ~600 files, 30 dirs     doc
```

Mit den übriggebliebenen, sprich beim Verschieben zurückgebliebenen, Beispielen kann man (nach jeweiligem Kopieren) beliebig experimentieren.

Die Datei gwt-servlet.jar verschiebt man aus dem Programme\GWT-Verzeichnis als „installed extensions“ in die Java-JDK-Installation, also etwa nach

```
C:\Programme\jdk\jre\lib\ext\
```

Damit funktioniert das Erstellen und Testen von AJAX-GWT-Servlets. Tomcat und Eclipse kennen bei richtiger Einbindung die betreffenden Pakete nun auch (wenn auch Letzteres manchmal erst nach Neusetzen derselben jdk-lib im Java-build-path).

Hinweis: Achtung Falle **!** Man kann die (Java-) GWT-Software — also alle übrigen .jar und .dll — nicht als „installed extensions“ der vorhandenen JDK-Installation zufügen (und damit u.A. auch leider nicht die vorgegebene „classpath-Wurstelei“ vermeiden).
Die GWT-Tools funktionieren dann einfach nicht.
Das gilt selbst, wenn dies „zusätzlich“ und nicht „anstatt“ geschieht.
Dieses Leiden teilt GWT mit vielen Apache-beeinflussten Java-Applikationen.
Eclipse und Tomcat hingegen genügen die "installed extensions" — hier also nichts mit classpath machen oder .jar-files zusätzlich in lib-Verzeichnisse kopieren.

In allen oben angeführten .cmd-Dateien nimmt man die folgende Änderung bzw. Ergänzung vor (am Beispiel applicationCreator.cmd):

```
@Echo.
@Echo G W T  application creator
@Echo.
@Echo ApplicationCreator %*
```

```
@Echo.
@java -cp "C:\programme\GWT\gwt-user.jar;
           C:\programme\GWT\gwt-dev-windows.jar"
           com.google.gwt.user.tools.ApplicationCreator %*
@Echo.
```

Mit den anderen .cmd-Dateien verfährt man sinngemäß und verschiebt sie alle in ein PATH-Verzeichnis, wie beispielsweise C:\bat\.

Abschließend kann man noch folgende zwei Maßnahmen ergreifen:

1.) Man kann die GWT-Dokumentation (nun) im Verzeichnis

`C:\Programme\GWT\doc\`

komplett komprimieren. Da fast alles .html ist, spart das etwa den halben Platz.

2.) Man kann im Dokumenten-Verzeichnis der JDK-Installation, also im Verzeichnis

`C:\Programme\jdk\docs\`

ein Unterverzeichnis

`C:\Programme\jdk\docs\googleGWTdocs\`

anlegen und dorthin die GWT-JavaDoc-Dokumentation, sprich Alles aus

`C:\Programme\GWT\doc\javadoc\`

kopieren bzw. verschieben. Dann hat man alle JavaDoc-Dokumentation an einem Fleck und kann bei eigener ebenfalls dort zu generierender JavaDoc einfacher relativ auf die GWT-JavaDoc verlinken.

Hinweis 1 zu neueren Versionen:

Bei Google Web Toolkit 1.4.60 sieht nach all den genannten Maßnahmen das Verzeichnis C:\programme\GWT etwa so aus:

```
27.08.2007  20:13             2.960 about.html
27.08.2007  20:13             1.031 about.txt
06.09.2007  17:13               213 applicationCreator.cmd
27.08.2007  16:07               237 benchmarkViewer.cmd
27.08.2007  20:13            12.460 COPYING
27.08.2007  20:11         3.599.580 gwt-benchmark-viewer.jar
27.08.2007  20:10        10.049.634 gwt-dev-windows.jar
27.08.2007  20:11            12.800 gwt-ll.dll
27.08.2007  20:13             5.013 gwt-module.dtd
27.08.2007  20:11         1.982.699 gwt-user.jar
06.09.2007  17:09               192 i18nCreator.cmd
06.09.2007  17:08               196 junitCreator.cmd
06.09.2007  17:10               201 projectCreator.cmd
27.08.2007  20:13            53.906 release_notes.html
08.08.2007  23:43           360.448 swt-win32-3235.dll
```

Beim Wechsel von einer älteren auf eine neuere Version für JDK und damit auch Tomcat müssen die Java-Quellen u.U. neu (nach JavaScript) übersetzt werden.

Umgekehrt gilt auch, dass beim Ausliefern (deploy; siehe unten) so neu übersetzten JavaScripts an einen anderen Tomcat in dessen JDK-Installation die servlet-api.jar gewechselt werden muss. Bei den beiden hier erwähnten GWT-Versionen 1.2.22 und 1.4.60 wäre das der Wechsel von

```
19.01.2007     17:01      276.629    gwt-servlet.jar
```

nach

```
27.08.2007     20:11      440.589    gwt-servlet.jar
```

oder (1.5.0 siehe Hinweise / Warnung am Anfang) dazu inkompatibel auch nach

```
23.05.2008      23:58     708.331    gwt-servlet.jar
```

Hierzu noch folgende Hinweise:

1. Ein solcher Wechsel geht nur wenn Tomcat und ggf. andere mit der JDK-Installation laufende Applikationen (kurz) gestoppt werden.
2. Alle GWT-Quellen für einen Web-Server (Tomcat) sollten immer gemeinsam übersetzt und ausgeliefert werden (wenn man ein Versionschaos zuverlässig vermeiden möchte).
3. Ein Symptom für unpassende servlet-api.jar bzw. gwt-servlet.jar zwischen Entwicklung/Übersetzung und Server sind bei der ersten Kommunikation mit dem Servlet abstürzende Kundenseiten, und im Tomcat-Log etwas wie

```
SCHWERWIEGEND: Exception while dispatching incoming RPC call
java.lang.SecurityException: Blocked attempt to access interface
'http://pd321s/serv-intra/java/gwt/',
which is either not implemented by this servlet or
which doesn't extend RemoteService;
this is either misconfiguration or a hack attempt
```

Im hier beschriebenen Versionskonflikt bezüglich gwt-servlet.jar ist es natürlich "miss-configuration" und nicht "hack attempt".

Hinweis 2 zu neueren Versionen — Falle: Bei einigen neueren Versionen des Google Web Toolkit (u.A. 1.4.60) setzt der GWT-Compiler einfach UTF8 als Kodierung der (.java) Quelldateien voraus. Dies entspricht meist nicht der Dateikodierung der Plattform, die bei europäischen (Windows-) Installationen i.A. ISO-8859-1 ist. Als Konsequenz funktioniert kein Text (Label, Button, etc.), der sich nicht auf primitives US-ASCII beschränkt.

Abhilfe schafft hier am besten das Umkodieren einer Kopie (!) jeder GWT-.java-Quelldatei unmittelbar vor dem Übersetzen, am Besten mit

```
java [de.a_weinert.apps.]UCopy -toUTF8 quellFile [ZielFile]
```

Das Werkzeug UCopy stellt diese Option ab Framework Version „weBib_03-80“ (August 2007) zur Verfügung.

Hinweis 3 zu neueren Versionen: Beim Wechsel von 1.4.50 nach 1.5.0 reicht das Alles — einschließlich des genannten Neuübersetzen + Stop/Start nicht. 1.5. bricht existierend Falle:

3. Test der Installation — das einfachste Hello modifiziert

Das mitgelieferte Hello-Beispiel wird auf seine Quellen reduziert, dabei wird das Paket von den „ellenlangen“ GWT-Paketen in ein eigenes — im Beispiel de.a_weinert.gwt.client — geändert. Die Hello- (Java-) Quelle wird mit GWT in eine AJAX-Seite übersetzt und in einem Browser und im so genannten GWT-hosted mode getestet.

Hinweis: Bei dem geringen Funktionsumfang des Hello-Beispiels sollte dieser Test-Einstieg in einem Zuge gelingen. Im Falle von Problemen muss man die Sache in ihre einzelnen Teil- und Modifikationsschritte aufspalten.

In ein Beispielverzeichnis Hello, das man auch gleich zum aktuellen Verzeichnis macht, erzeugt man folgenden Verzeichnisstruktur und kopiert (nur!) die dargestellten Quell- und Script-Dateien aus dem GWT-Hello-Beispiel dorthin:

```
│
└──Hello\              <aktuelles Verzeichnis:>
     │    Hello-compile.cmd
     │    Hello-shell.cmd
     │
     └────de\
          └──a_weinert\
               └──gwt\
                    └─hello\
                         │   Hello.gwt.xml
                         │
                         ├──client\
                         │         Hello.java
                         │
                         └──public\
                                   Hello.html
```

In der Datei hello.java ändert man lediglich die package-Anweisung:

```
package de.a_weinert.gwt.hello.client;
```

Die Datei hello.html ändert man in:

```
<html>
   <head>
      <meta name='gwt:module'
          content='de.a_weinert.gwt.hello.Hello'>
   <title>Hello (package de.a_weinert...)</title>
   </head>
   <body bgcolor="white">
      <script language="javascript" src="gwt.js" />
   </body>
</html>
```

Die Datei hello.gwt.xml modifiziert man so:

```
<module>
   <inherits name="com.google.gwt.user.User"/>
   <entry-point class="de.a_weinert.gwt.hello.client.Hello"/>
</module>
```

Die Dateien Hello-compile.cmd bzw. Hello-shell.cmd ändert man so (Hello-compile.cmd):

```
java -cp
    "%~dp0;C:\programme\GWT\gwt-user.jar;
     C:\programme\GWT\gwt-dev-windows.jar"
        com.google.gwt.dev.GWTCompiler
         -out "%~dp0\www" %* de.a_weinert.gwt.hello.Hello
```

bzw. (Hello-shell.cmd)

```
java -cp
   "%~dp0;C:\programme\GWT\gwt-user.jar;
    C:\programme\GWT\gwt-dev-windows.jar"
        com.google.gwt.dev.GWTShell
        -out "%~dp0\www"  %*
             de.a_weinert.gwt.hello.Hello/Hello.html
```

Hinweis: Beide sind (lange) Einzeiler. Vergleiche auch oben die nach C:\bat\ verschobenen .cmd-Dateien.

Laufen Lassen von Hello-compile.cmd führt im fehlerfreien Fall zur Meldung

```
Output will be written into
   E:\Tests\....\Hello\www\de.a_weinert.gwt.hello.Hello
Copying all files found on public path
Compilation succeeded
```

Dabei entstehen im Arbeitsverzeichnis hello zwei neue Unterverzeichnisse

```
23.01.2007    11:11    <DIR>    .gwt-cache
23.01.2007    11:11    <DIR>    www
```

mit zahlreichen Dateien. In einem Unterverzeichnis

```
E:\Tests\....\Hello\www\de.a_weinert.gwt.hello.Hello
```

wurde nun eine Datei hello.html generiert, welche die eigentliche Hello-AJAX-Seite ist: Anklicken und Freuen!

Das Laufen Lassen von Hello-shell.cmd führt in eine graphische Oberfläche (hosted mode — nur für Tests ohne „echten“ Tomcat) und erzeugt vorher ein neues Unterverzeichnis

```
23.01.2007    11:38    <DIR>    tomcat
```

mit minimalen Tomcat-Konfigurationsdateien (für den embedded Tomcat).

Im Erfolgsfalle sieht man die generierte AJAX-Seite in der graphischen Umgebung und über Port 8888 von einem dort eingebundenen minimalen Tomcat geliefert; siehe Bild 1.

Dies simple Hello-Beispiel diente auch nur zum minimalen kennen (und laufen) Lernen und zum Test der GWT-Installation

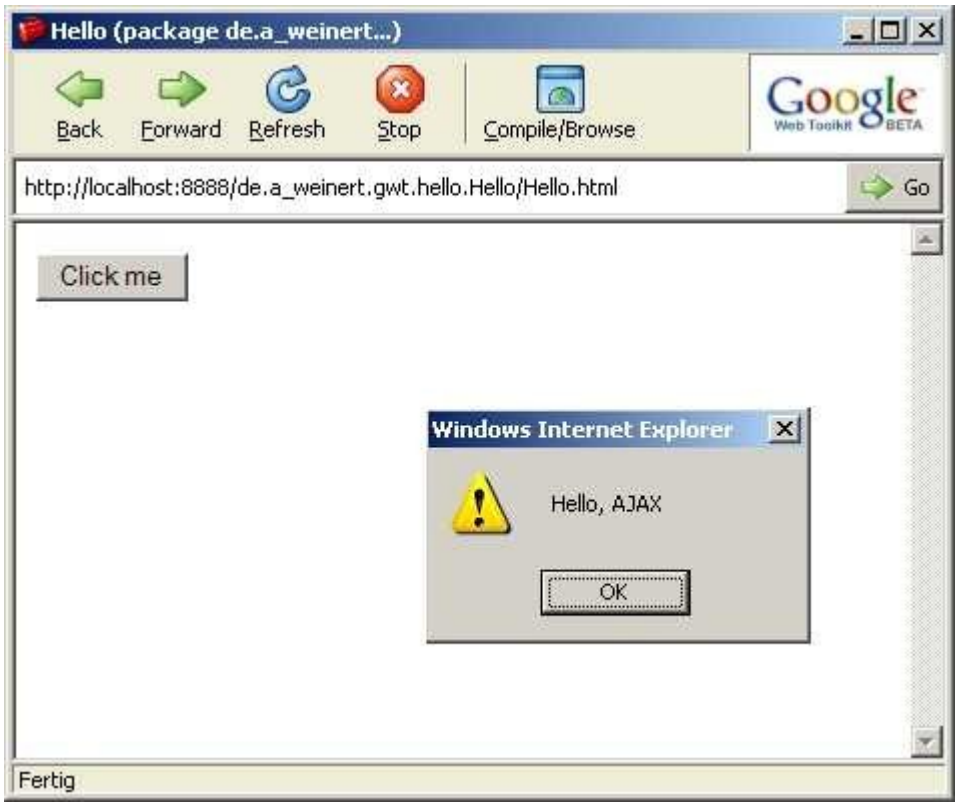

Bild 1: Hello im Google hosted mode

Hinweis, Falle: Achtung, einige Änderungen der zugrundeliegenden .html und .xml-Dateien im Sinne von besserem (x)html (<meta blah blah blah /> z.B. oder Leerzeichen statt Tabulatoren) scheinen die Lauffähigkeit im hosted mode zu beeinträchtigen.
(ToDo: gelegentlich klären; andererseits ist der „hosted mode“ herzlich uninteressant.)

4. Tools — Tomcat- und Web-Deployment

Das mitgelieferte und oben modifizierte Script `applicationCreator.cmd` erzeugt "aus dem Bauch" ein weiteres Hello-Beispiel. Es ist als Ausgangspunkt für eigene Arbeiten gedacht, demonstriert aber auch ein paar über das obige Einfachstbeispiel hinausgehende Möglichkeiten.

Mit

```
   d:
   md D:\tmp\projectCreate
   cd D:\tmp\projectCreate
   applicationCreator de.a_weinert.gwt.samples.client.Hello
...pause Gucken, was da schon entstanden ist
   Hello-compile
   Hello-shell
```

erzeugt man in einem leeren (Spiel-) Verzeichnis ein Projekt mit Quellen und Scripts, das man mit dem letzten beiden Befehlen auch gleich übersetzt und im hosted mode anschaut. Bis auf ein paar neue Aspekte, entspricht dies dem vorangegangenen Kapitel.

Für das weitere wird ein gemäß [5] eingerichteter Tomcat als J2EE-Container vorausgesetzt.

Den Inhalt des durch das obige Kommando Hello-compile entstandenen Unterverzeichnisses ...\www, sprich das gesamte Verzeichnis

```
D:\tmp\projectCreate\www\de.a_weinert.gwt.samples.Hello
```

wird in das Web-Applikationsverzeichnis des Tomcat kopiert also nach etwas wie

```
D:\......\webAppsDev.
```

In einer dortigen Verweisseite (i.A. index.html) ergänzt man zum bequemen Testen etwas wie

```
<li><a href="de.a_weinert.gwt.samples.Hello/Hello.html">
        Simples Hello AJAX <br /></a></li>
```

Über diesen Link kann man nun ganz normal via http (Port 80) über den bereits betriebenen Tomcat-WWW-Server auf das gerade generierte AJAX-Hello zugreifen.

Unter der Annahme, dass dies Hello zu etwas Vertraulichem weiterentwickelt wird, soll nun noch der Zugriff über SSL bzw. https erzwungen werden.

In Tomcat's Grundkonfigurationsdatei, i.A.

```
C:\Programme\Apache\Tomcat 5.5\conf\web.xml
```

wird hierzu an den betreffenden Stelle (ziemlich weit hinten, siehe [5]) folgendes ergänzt

```
<!--  25.01.2007 14:53 AJAX GWT - Test -->
   <security-constraint>
    <web-resource-collection>
      <web-resource-name>AJAX Test</web-resource-name>
      <url-pattern>/de.a_weinert.gwt.samples.Hello/*</url-pattern>
    </web-resource-collection>
    <user-data-constraint>
       <transport-guarantee>CONFIDENTIAL</transport-guarantee>
    </user-data-constraint>
   </security-constraint>
```

Das war's. Spätestens nach einem Tomcat-Neustart ist diese Hello-AJAX-Seite automatisch und nur noch über SSL zugreifbar. Mit sinngemäß etwas Zusätzlichem wie

```
      <auth-constraint>
         <role-name>domain_group</role-name>
      </auth-constraint>
```

macht man auch noch die Authentifizierung eines Nutzern aus der angegebenen Gruppe (Rolle) zur Voraussetzung; vgl. wieder [5].

Dieses gerade generierte zweite Hello-Beispiel und die geschilderte Web-/Tomcat-Verbreitung wird man natürlich so nicht nutzen, sondern als Ausgangspunkt für sinnvollere AJAX-Anwendungen nehmen. Dabei wird man die Quellen i.A. in vorhandene Eclipse-Projekte einbeziehen.

Stackrechner — zum Ersten

Ein ausgeführtes entsprechendes Beispiel (mit Quellen) finden Sie unter a-weinert.de/java/gwt/stack_calc.html bzw. unter a-weinert.de/java/ (Bild 2).

Hier sind noch folgende Hinweise zu Namen und Verzeichnisbäumen nützlich:

- Wie schon beim ganz einfachen Hello-Beispiel des letzten Kapitels geschehen, können Sie das generierte (Zwischen-) Verzeichnis `.../src/...` einfach weglassen bzw. aus dem Baum rausschneiden. Damit wird die Integration in Eclipse-Quellbäume und vernünftige Paketstrukturen einfacher.
- Die generierte Start-html-Datei darf beliebig heißen.
- Das gemäß dem verwendeten Paketnamen komplex benannte Zielverzeichnis der Übersetzung (das an das angegebene „build-Ziel“ angehängt wird) ist so nicht erforderlich. Das Übersetzungsergebnis kann in ein beliebig benanntes Verzeichnis geschoben werden.
- Von den bei der Übersetzung generierten xml-, html- und Bild-Dateien sind i.A. nicht alle für die Zielanwendung und den / die Ziel-Browser erforderlich. Falls man das Laden unnötiger Dateien auf den Webserver aus Platzgründen vermeiden vermeiden muss, kann man die nötige Untermenge allerdings nur experimentell klären: Dabei geht man dann von sinngemäß lediglich den folgenden Dateien aus und befriedigt sukzessive die Browser-Fehlermeldungen.

```
02.02.2007  15:02     2.635   de.a_weinert.gwt.NAME.nocache.html
02.02.2007  14:57    17.374   gwt.js
02.02.2007  14:57       464   history.html
02.02.2007  14:57     6.420   stack_calc.html
```

  Dabei wird sich auf jeden Fall herausstellen, dass mindestens eine der nun wirklich „pervers“ benannten, generierten JavaScript-Dateien fehlt. Deren Namen sehen in etwa so aus:

```
ACD888265A1BDE3CF916827A44F27B2F.cache.html
```

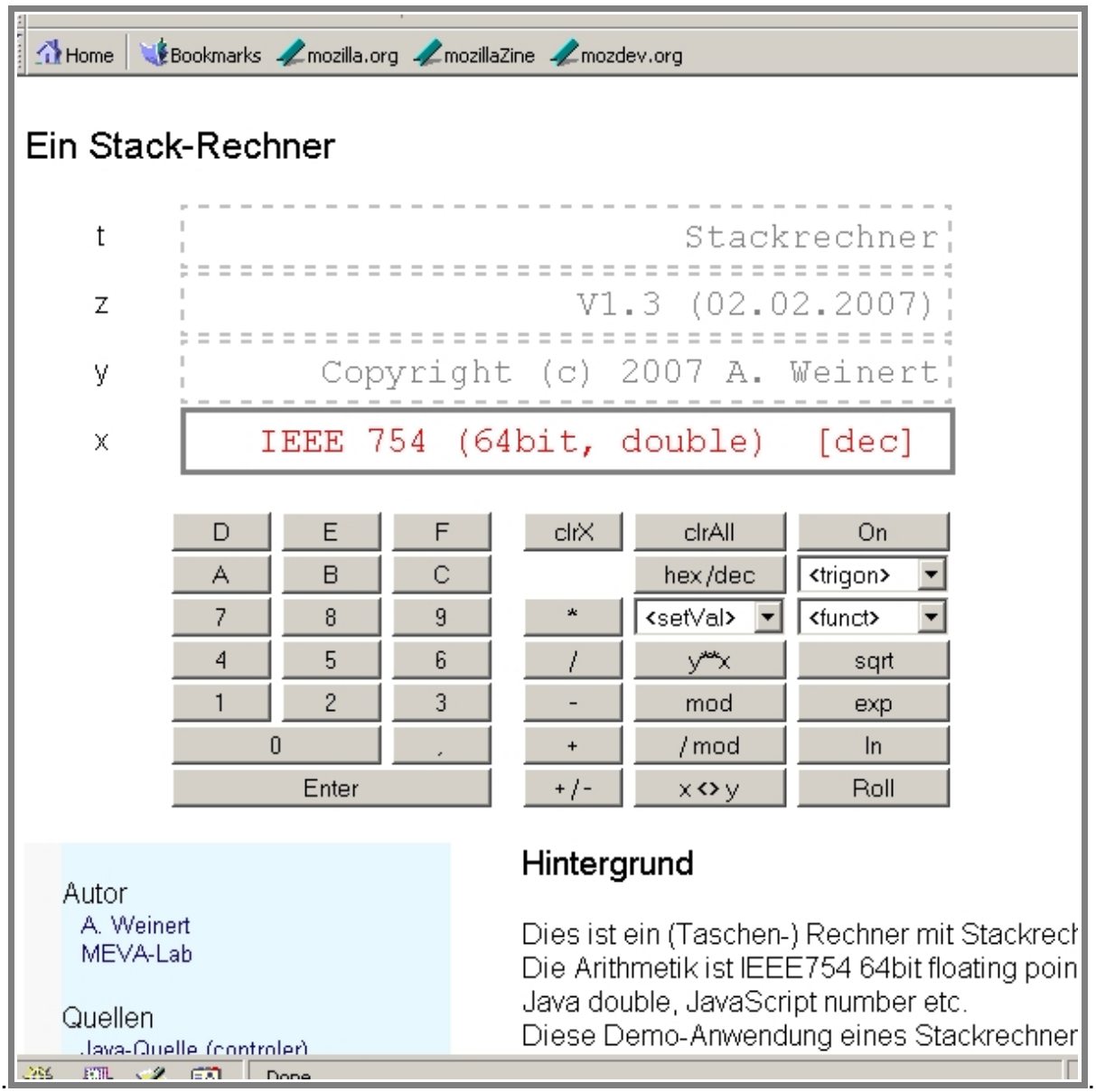

Bild 2: Ein AJAX-GWT-Stack-Rechner.

- Absolut störend ist nun leider, dass die kleinste Änderung an einer Java-Quelle — und da reicht schon das Ersetzen von cvsNT-Tags durch Klartext — diese perverse Benennung unkontrollierbar ändert.

(ToDo: Klären, ob hier irgendwie Einfluss/Kontrolle möglich ist.)

Achtung: Solche Namenszufälligkeiten sind nun für ein automatisches Script-gesteuertes Deployment, eine Versionskontrolle und einiges Andere recht störend. Dass sich Verzeichnisse auf Webservern so mit „ABCD.cache.html-Leichen“ füllen mag noch das geringste Übel sein.

5. Das Web-Dienst-Einstiegsbeispiel

Hinweis: Das Nachfolgende setzt einen J2EE-Container voraus. Ein einfacher Webserver (Apache) genügt nicht mehr. I.A. wird man einen Tomcat einsetzen.
Eine geeignete Installation gemäß [5] wird spätestens nun erforderlich.

Das eben verwendete Rechnerbeispiel wird zu einem Web-Dienst gemacht. Anstatt dass der Automat bzw. das model (von Java nach JavaScript GWT-übersetzt) auf dem Client im Browser läuft, soll es als Java-Servlet in einem J2EE-Container (Tomcat) auf dem Server laufen. Die mit GWT-übersetzen Client-Seite kommuniziert mit diesem Servlet via AJAX. Einen solchen Taschenrechner auf dem Server laufen zu lassen, ist an sich sinnlos; es dient als Einstiegsbeispiel für einen nicht ganz trivialen Web-Service mit GWT.

Um den Java-/JavaScript-Unterschied zu zeigen, ist es nun aber ein reiner ganzzahliger Rechner (64 Bit, 2er-Komplement, long), was mit der nicht typisierten Sprache JavaScript so gar nicht darstellbar ist; Bild 3. Die Quellen finden Sie wieder unter a-weinert.de/java/.

Die nachfolgend genannten Datei- und Verzeichnisnamen beziehen sich auf genau dieses Beispiel bzw. auf eine sinnvolle Standardinstallation von Tomcat und Java-Werkzeugen.

5.1 Die Programmierung

Die Schritte für die Java-Programmierung (in Eclipse) sind folgende:

1.) Schreiben des Servlets, d.h. des Arbeitskodes für die Serverseite, z.B. RemoteCalcMod.java (Automat, model im Rechnerbeispiel).
Die Klasse muss `com.google.gwt.user.server.rpc.RemoteServiceServlet` beerben und das in Schritt 2 hergestellte Interface implementieren.
Eclipses „extract interface“ erledigt letzteres gleich mit.
Hinweis: Wegen eines Tomcat-Fehlers muss diese Klasse als Startklasse eines Servlets leider in das anonyme Paket; die anderen Klassen kommen selbstverständlich in die vom Projekt her gegebenen Pakete.

2.) Herstellen des Interfaces des Dienstes RemoteCalcIntf.java. (im Beispiel).
Das Interface kann in Eclipse mit „extract interface“ aus der eben erstellten Service-Klasse mit ganz wenigen Klicks erstellt werden. Das so erstellte Interface muss `com.google.gwt.user.client.rpc.RemoteService` erweitern bzw. beerben. Dies ist also entsprechend zu ergänzen.

3.) Herstellen des „asynchronen“ Interfaces.
Dies geschieht einfach als Kopie durch „Speichern unter“, Umbenennen des Typs und Weglassen des Beerbens von `.RemoteService.`
Im Beispiel entstünde so RemoteCalcIntf.java.
Anschließend sind alle Methodensignaturen folgendermaßen zu ändern:
- Rückgabe void und
- zusätzlicher letzter Parameter vom Typ AsyncCallback, (genauer `com.google.gwt.user.client.rpc.AsyncCallback`).

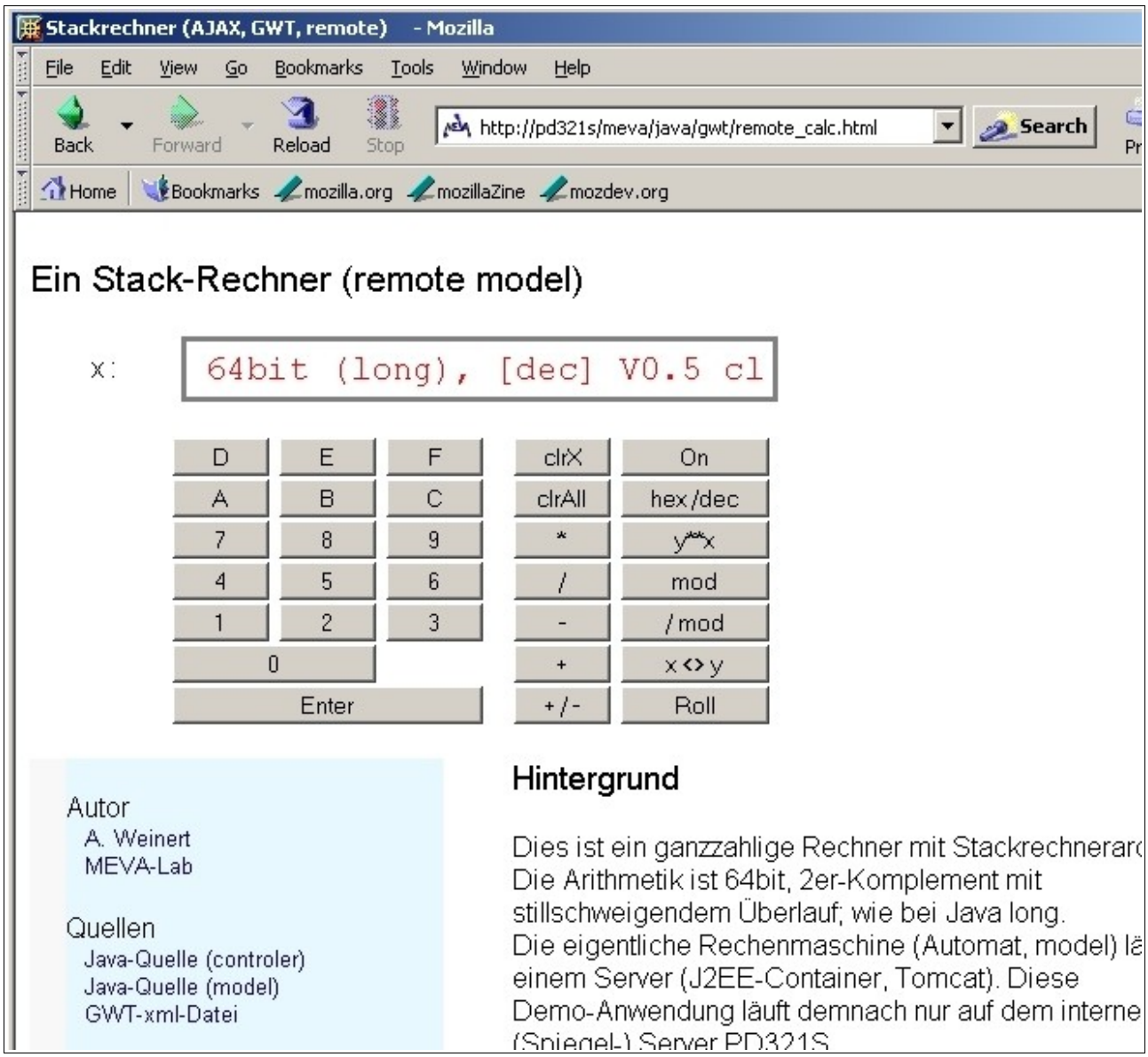

Bild 3: Ein AJAX-GWT-Stack-Rechner als Web-Service (long).

Die Signaturumstellung geschieht einfach durch folgende („replace all“) Ersetzungen:

abstract String	→	abstract void	(sinngemäß für jeden Rückgabetyp),
);	→	, AsyncCallback cB);	
(, AsyncCallback cB);	→	(AsyncCallback cB);	

Die folgenden Schritte für die Programmierung der Client-Seite entsprechen vollkommen denen des Beispiels aus dem vorangegangenen Kapitel (reiner GWT-Rechner, Bild 2).

4.) Erstellen des clientseitigen Controllers RemoteCalcGWT (im Beispiel).
Die Klasse muss `com.google.gwt.core.client.EntryPoint` beerben und wird mit dem GWT-Compiler in JavaScript übersetzt.

5.) Erstellen der GWT-Steuerdatei RemoteCalcGWT.gwt.xml mit folgendem Inhalt:

```
<!--
```

```
 * Stackrechner mit AJAX / GWT
 * (c) 2007 Albrecht Weinert
 * Version $Revision: 1.1 $ ($Date: 2007/02/08 15:53:12 $)
-->
<module>
     <inherits name='com.google.gwt.user.User'/>
     <entry-point class='de.a_weinert.gwt.client.RemoteCalcGWT'/>
     <servlet     class='RemoteCalcMod'
                          path='/webappsdev/servlet/RemoteCalc' />
</module>
```

6.) Erstellen der darstellenden HTML-Seite remote_calc.html mit folgenden-Zeilen

```
<meta name='gwt:module'
         content='de.a_weinert.gwt.RemoteCalcGWT' />
</head><body class="body" >
<script language="javascript" src="gwt.js"></script>
<iframe id="__gwt_historyFrame"
         style="width:0;height:0;border:0"></iframe>
```

Hinweis, Alle clientseitigen Klassen, also die die wie bereits beschrieben vom GWT-Compiler in JavaScript übersetzt werden, und alle eben genanten Interfaces müssen in einem Paket sein.
Aber die im eben genannten Schritt 1 erstellte Server-Klasse muss in einem anderen Paket sein. Und, wie bereits erwähnt, muss dies für bestimmte Tomcat-Versionen das anonyme Paket sein. Um das Übersetzen und das (Tomcat-) Deployment dieses Servlets kümmert sich das GWT nicht.

Zur Übersicht zeigt Bild 4 die Verzeichnis- und Paketstruktur der erwähnten Beispielquellen. Das rein clientseitige Beispiel des vorangegangenen Kapitels („Stack..“) ist in dieser Darstellung mit dabei.

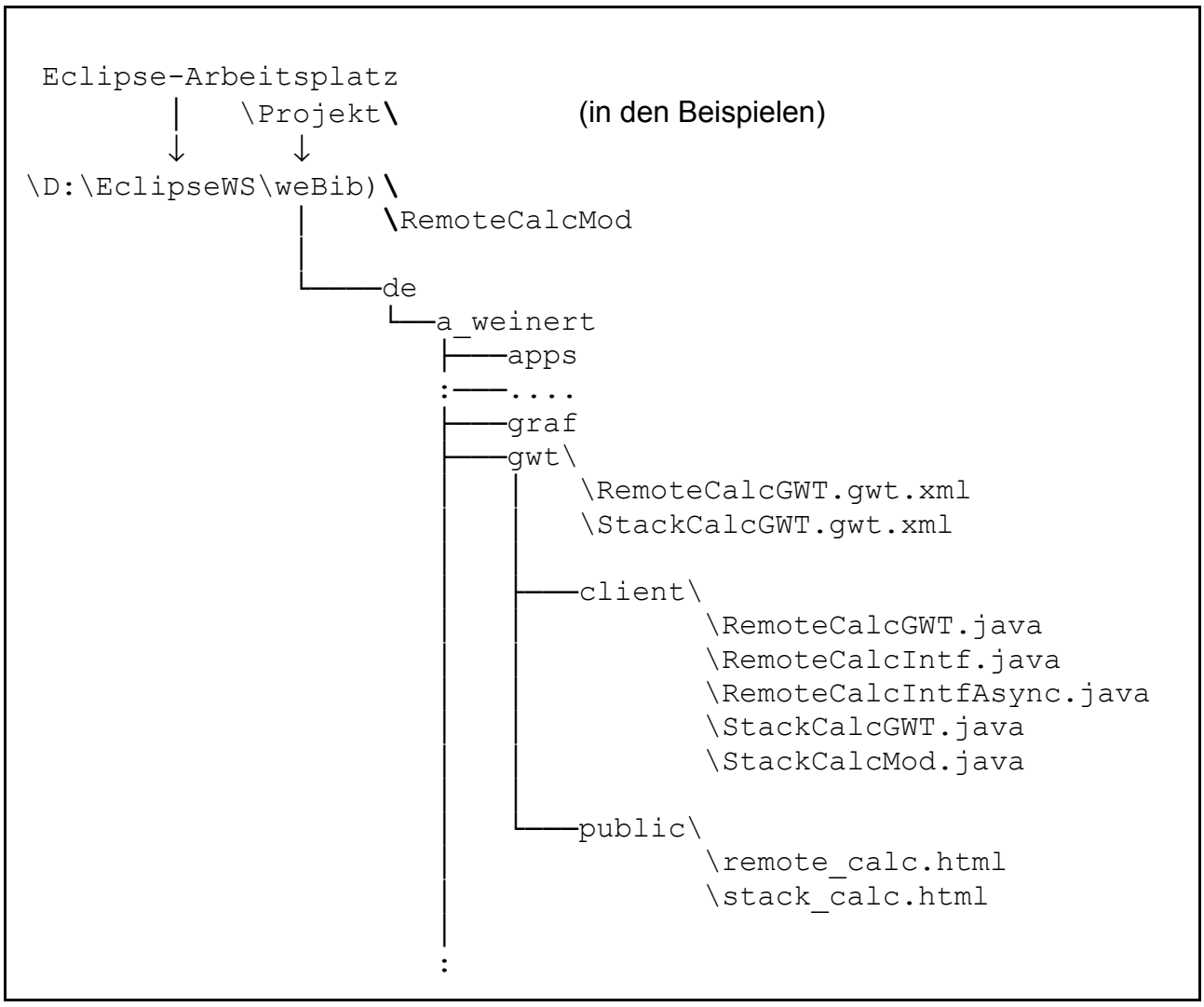

Bild 4: Die Verzeichnis- und Paketstruktur (der Beispielquellen).

Die Java-Quellen für die Client- bzw. Browser-Seite werden werden mit dem GWT-Compiler in JavaScript übersetzt und zusammen mit der zugehörigen HTML-Seite als statischer Content in ein Lieferverzeichnis eines Web-Servers gebracht. Ein entsprechendes Script sieht im Auszug sinngemäß so aus:

```
java Del -empty build\www\
md build\www
java -cp
  ".\;C:/Programme/GWT/gwt-user.jar;C:/Programme/GWT/gwt-dev-
windows.jar"  com.google.gwt.dev.GWTCompiler -out "build\www"
    %* de.a_weinert.gwt.RemoteCalcGWT
@if ERRORLEVEL 1 goto :error

@Echo Dest.(Tomcat)  workNoSave\WWWBaustelle\meva-lab\java\gwt
@Echo clear GWT-generated, GWT-named files
@Echo.
del  D:\workNoSave\WWWBaustelle\meva-lab\java\gwt\*.cache.html
```

```
del  D:\workNoSave\WWWBaustelle\meva-lab\java\gwt\*.cache.xml
@Echo.
@Echo uppdate newly build, copied files  -------------
@Echo.
java Update build\www\de.a_weinert.gwt.StackCalcGWT\+.+
            D:\workNoSave\WWWBaustelle\meva-lab\java\gwt\

java Update build\www\de.a_weinert.gwt.RemoteCalcGWT\+.+
              D:\workNoSave\WWWBaustelle\meva-lab\java\gwt\
```

Das Programm für die Serverseite ist ein Servlet. Es wird ganz normal mit javac übersetzt und letztlich genau wie jedes andere Servlet in einen J2EE-Container, sprich i.A. Tomcat, gestellt. Siehe dazu [5] und die folgenden Abschnitte.

Die AJAX-XML-Kopplung der Client-Anforderungen an die Methoden dieses Servlets übernimmt (neben einigem Anderen und dankenswerterweise) die GWT-Mechanik nach bestimmten vorgegebenen (Entwurfs-) Mustern, die einzuhalten sind. Für das (runterladbare) Beispiel sind diese aus dem Klassendiagramm, Bild 5, erkennbar.

5.2 Vorbereitung von Tomcat für GWT-Web-Services

Das (GWT-AJAX-) Servlet beerbt `javax.servlet.http.HttpServlet` nicht direkt, sondern indirekt über `com.google.gwt.user.server.rpc.RemoteServiceServlet`; vgl. Bild 5.

Die (einfache) Konsequenz ist, dass in der Tomcat-Installation diese Dateien

```
9.01.2007  17:01        276.629   gwt-servlet.jar
9.01.2007  17:01      1.146.806   gwt-user.jar
```

zu ergänzen sind. Sprich sie sind einfach aus der GWT-Installation nach

```
C:\Programme\Apache\Tomcat 5.5\common\lib
```

zu kopieren.

Ein Neustart von Tomcat ist nach einer solchen Ergänzung i.A. erforderlich.

Ohne diese Grundergänzung geht überhaupt kein GWT-Servlet — unabhängig davon muss auch das Deployment des einzelnen Servlets stimmen, siehe nächster Abschnitt.

Ergänzt man im selben genannten Verzeichnis auch die (möglichst jeweils neueste)

```
08.02.2007  17:19       781.308   aWeinertBib.jar
```

so stehen auch den GWT-Servlets, genau wie den „normalen“ HTML liefernden, alle Möglichkeiten des Frameworks zur Verfügung.

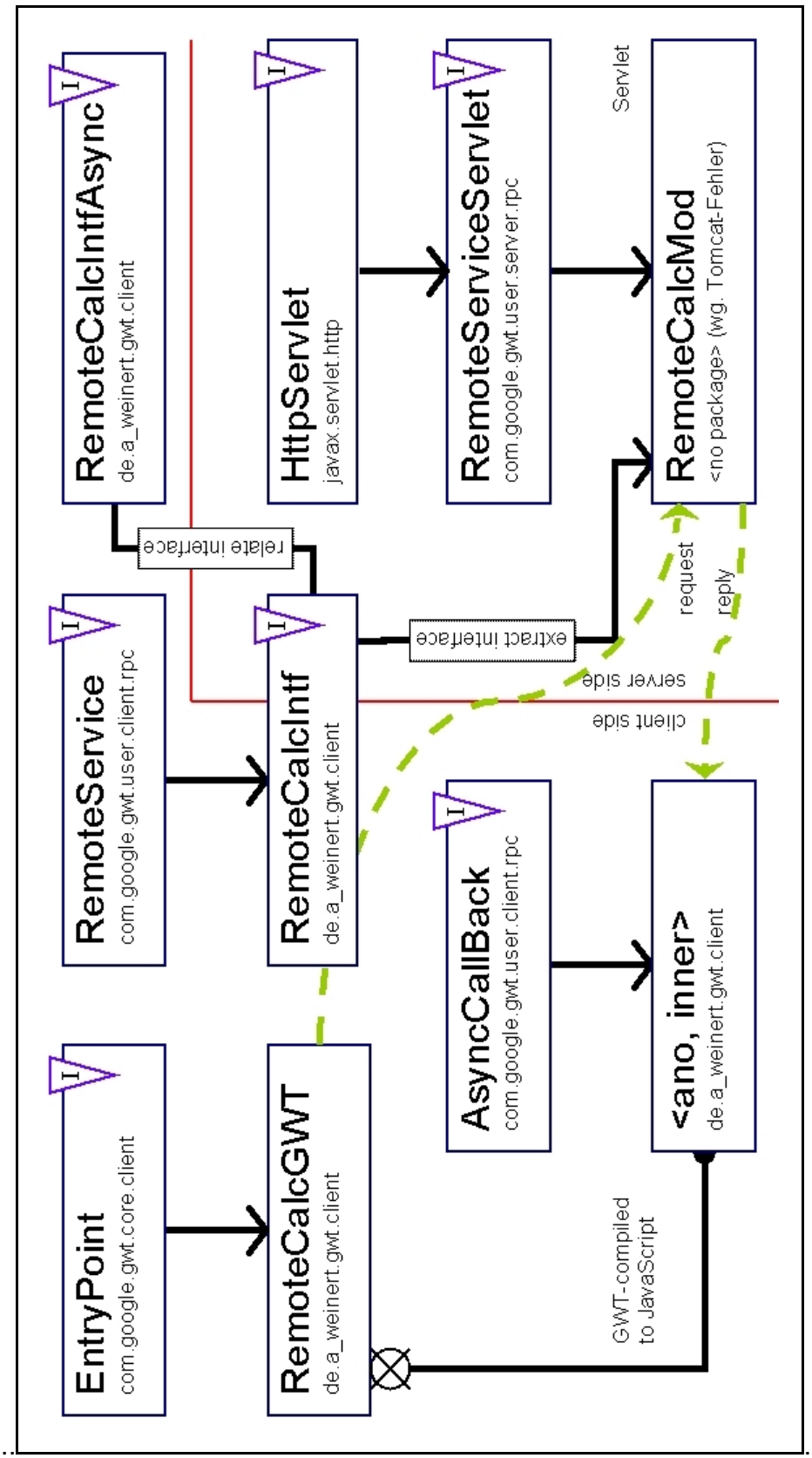

Bild 5: Klassendiagramm zum Web-Dienst-Rechnerbeispiel.

5.3 Tomcat-Deployment des Servlets

Die Voraussetzung, dass das Servlet überhaupt angesprochen wird, ist das bereits (mehrfach) beschriebene Deployment des statischen Content, also der einbettenden HTML-Seite und der mit dem GWT-Compiler übersetzten zugehörigen JavaScript-Seiten. Ist dies korrekt geschehen, liefert in einem Browser (Mozilla) eine URL wie

```
http://pd321s/meva/java/gwt/remote_calc.html
```

(mit Ihren Verhältnissen angepasstem Servernamen) genau das in Bild 3 weiter oben gezeigte Startbild. Dazu braucht man noch kein Servlet.

Fall: Static content „geht nicht“

Falls schon dies nicht geht, bitte zurück nach oben, nach [5]. Sinnvoll ist es hier die Fälle

„wird nicht geliefert“ und
„wird geliefert, sieht aber nicht wie gewollt aus“

zu unterscheiden. In letzterem Fall muss man Quellen ändern, neu übersetzen etc..

Im ersteren Falle mache man sich auch im Hinblick auf das Folgende die Tomcat- “Liefermechanik“ klar, und dass es hier teilweise auf jeden Buchstaben inklusive G/K-Schreibung ankommt. Im obigen Beispiel

```
http://pd321s/meva/java/gwt/remote_calc.html
```

gilt folgendes:

- `http://pd321s/`

bezeichnet das Protokoll, den Tomcat-Webserver (ohne Angabe Port 80 als default). Dies Alles muss da sein, laufen, (DNS-) aufgelöst werden etc.
Wenn schon das nicht geht, „.. forget the rest“ (wie immer).

- `..../meva/`

bezeichnet eine gleichnamige XML-Konfigurationsdatei des angesprochenen Tomcat.
Im Beispielfalle wäre es diese:

```
C:\Programme\Apache\Tomcat 5.5\conf\Catalina\localhost\meva.xml
```

Diese Datei wiederum enthält das Startverzeichnis des betreffenden statischen Inhalts.
Und n.b.: mit „meVA“ wird eine MEva.xml nicht unbedingt gefunden, auch nicht von einem Tomcat unter Windows.

- `.../java/gwt/remote_calc.html`

ist nun endliche eine relative Dateiangabe bezogen auf das in der genannten XML-Datei (als Attribut docBase) angegebene Verzeichnis. Auch hier kann trotz Basis=Windows G/K-Schreibung relevant sein.

Hinweis, Wenn man dies „verinnerlicht“ und „hinbekommen“ hat, mache man sich klar, dass vom Tomcat, die Servlets (Servlet-Klassen, -Ressourcen und was noch dranhängen mag) nach einem gleichen Schema gefunden werden.

Fall: Static content geht, aber nicht das Servlet

Dieser Fall liegt vor, wenn

```
http://pd321s/meva/java/gwt/remote_calc.html
```

zu dem Startbild (Bild 3) führt, aber das erste Drücken einer Taste nicht die erhoffte, sprich auch im Servlet programmierte Taschenrechnerreaktion liefert.

Meist ärgert dann etwas wie

> „The requested resource (/webAppsDev....RemoteCalcMod) is not available“

in der (ellenlangen) Antwort mit einem zugehörigen Eintrag im Tomcat-Log `s la:

```
...SCHWERWIEGEND: Error loading WebappClassLoader
  delegate: false
  repositories:    /WEB-INF/classes/
    ----------> Parent Classloader:
    org.apache.catalina.loader.StandardClassLoader@1777b1
                 ............RemoteCalcMod
   java.lang.ClassNotFoundException:
                ......RemoteCalcMod
     at ........
```

Langer Rede kurzer Sinn:
Klassen und oder Ressourcen, die zum (AJAX-) benutzten Servlet gehören, werden nicht gefunden. So einfach das klingt, so frustrierend ist in solchen Fällen die Fehlersuche.

Letztlich muss man sich dann an der — hinreichend verstandenen — Tomcat- (bzw. J2EE-) „Auflösungskette“ für ein Servlet „langhangeln“ und jeden Schritt kontrollieren.

Zunächst aber noch mal die bereits erwähnten Standard-Fehler vorweg:

1. Stimmt haargenau die Groß-/Kleinschreibung (siehe eben)?
2. Ist die Start-Servlet-Klasse im anonymen Paket (wg. Fehler von mindestens Tomcat 5.5)?

Fall: Servlet wird noch nicht gefunden

Wenn diese Fragen nicht die Lösung brachten (und ein oben genannter Versionskonflikt auszuschließen ist), dann also durch **. . . .**

Die Tomcat- bzw. J2EE- Servlet - „Auflösungskette“

Wenn sonst alles richtig programmiert ist, führt der erste Druck auf eine Knopf in der Startseite (Bild 3) zum erstmaligen Laden des angesprochenen Servlets.

Im (nun einfach weiter verwendeten Beispiel) steht die Bezeichnung des Servlets

a) in der Datei

```
D:\EclipseWS\weBib\de\a_weinert\gwt\RemoteCalcGWT.gwt.xml
```

als

```
<servlet     class='RemoteCalcMod'
                      path='/webappsdev/servlet/RemoteCalc' />
```

und

b) in der Quelle

```
D:\EclipseWS\weBib\de\a_weinert\gwt\client\RemoteCalcGWT.java
```

als

```
leCalc =
   (RemoteCalcIntfAsync ) GWT.create(RemoteCalcIntf.class);

((ServiceDefTarget)leCalc).setServiceEntryPoint(
                          "/webappsdev/servlet/RemoteCalc");
```

in der Methode onModuleLoad().

Auch wenn der Verdacht naheliegt, dass eine der beiden Angaben überflüssig sein könnte: Beide Angaben sollten erst mal Buchstabe für Buchstabe übereinstimmen.

Die Auflösung dieser Angabe ist nun noch ein bis zwei Stufen komplizierter als (vgl. oben) für den „static content“, wenn auch der erste Schritt übereinstimmt. Von der Angabe

```
webappsdev/servlet/RemoteCalc
```

bezeichnet

```
webappsdev
```

indirekt über

```
C:\Programme\Apache\Tomcat 5.5\conf\Catalina\localhost\webappsdev.xml
```

das Tomcat-lokale Verzeichnis für die Dateien des Web-Dienstes. Im konkreten Beispiel des Servers PD321S wäre das

```
...D:\www\webAppsDev
```

Ein solches Web-Service-Verzeichnis hat ein ganz klar vorgegebene Struktur. Bild 6 zeigt diese — hier dargestellt für den Zustand ausgepackt ohne .jar/.war — konkretisiert anhand der hier zugrundegelegten Beispiele; siehe auch [5].

Zunächst müssen (vgl. immer Bild 6) die Servlet-Klassen und alle weiteren Klassen und Ressourcen, die sie benötigen in das Verzeichnis

```
WEB_INF\classes\
```

Im Beispielfall kommt die Datei

```
08.02.2007  17:28         3.530    RemoteCalcMod.class
```

direkt dorthin und die implementierte Interface-Klasse

```
08.02.2007  17:28           499    RemoteCalcIntf.class
```

in Unterverzeichnisse entsprechend ihrem Paket.

Nochmal: Servlet-Startklassen müssen bei Tomcat (5.5) leider ins anonyme Paket!

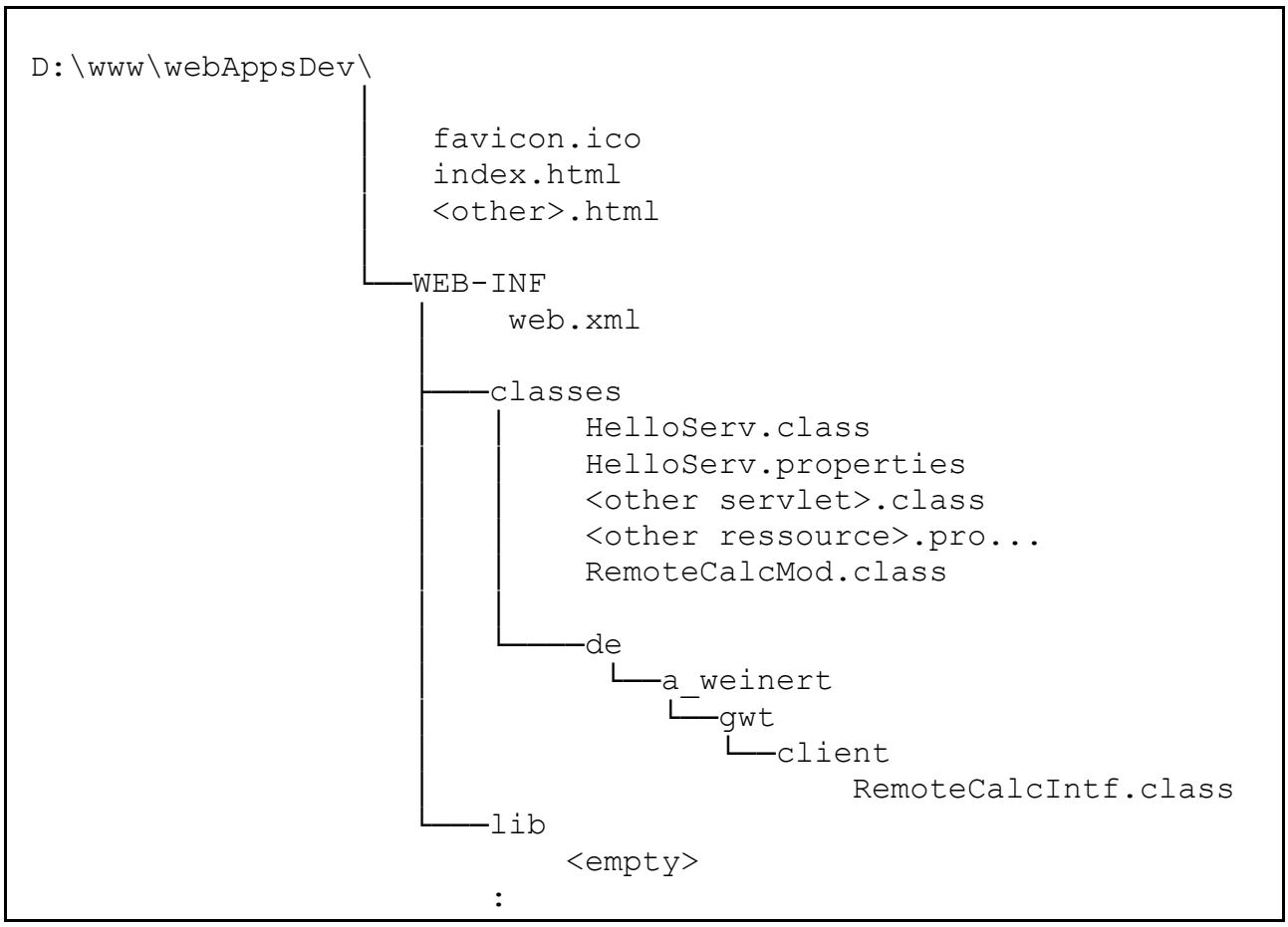

Bild 6: Die Webservice-Verzeichnisstruktur (konkretisiert anhand der Beispiele).

Entscheidend für das Finden des Servlets ist nun die Datei

```
09.02.2007  08:29         4.276    WEB-INF\web.xml
```

und hierin nun speziell folgende zwei Einträge:

```
<!-- AJAX mit Web-Dienst (servlet); erster Test:  07.02..... -->
   <servlet>
      <servlet-name>RemoteCalc</servlet-name>
      <servlet-class>RemoteCalcMod</servlet-class>
   </servlet>
   <servlet-mapping>
      <servlet-name>RemoteCalc</servlet-name>
      <url-pattern>/servlet/RemoteCalc</url-pattern>
   </servlet-mapping>
```

Von der Angabe

```
webappsdev/servlet/RemoteCalc
```

(aus u.A. `setServiceEntryPoint()`; vgl. oben) bezeichnet nun also der Teil

```
/servlet/RemoteCalc
```

zweifach indirekt über

```
                servlet-mapping          servlet
                      |                    |
    url-pattern   →   servlet-name   →   servlet-class
```

das Servlet.

Wenn dies alles — bis hin zu den Feinheiten der Groß-/Kleinschreibung — stimmt, liefert das nachvollzogene und in einen Tomcat gestellte Beispiel einen bedienbaren, funktionierenden Taschenrechner als Web-Dienst mit GWT-AJAX.

Was noch zu tun bleibt

Für den Fall, dass man einen solchen Taschenrechner (oder sinngemäß eine Simulation, Demo etc.) so als Web-Dienst anbieten möchten, braucht man noch ein

- Sitzungsmanagement

für die Speicherung des Zustands des Automaten (model). Ein solches Sitzungsmanagement ist für Java-Servlets eine vergleichsweise leichte Übung (vgl. RemoteCalcMod.java).

Für den Fall der Bedienung und Beobachtung (B&B) eines Prozesses oder einer Anlage als (GWT-AJAX-) Web-Dienst vermittelt durch ein Servlet benötigt man ein solches Sitzungsmanagement oft nicht, da der eine gemeinsame Zustand auch bei vielen gleichzeitigen Bedienern und Beobachtern in der Anlage bzw. ihrer Steuerung-/Automatisierungsanwendung steckt.

Was man für solche B&B-Anwendungen i.A. aber unbedingt braucht. ist

- Protokoll- und (i.A. SSL)
- Zugriffskontrolle (Authentifizierung).

Beides funktioniert — auch mit GWT-AJAX — mit den zugehörigen Tomcat-Verfahren, die per se auch nicht ganz einfach zu beherrschen sind; vgl. auch [5]. Im Zusammenhang mit einem à la Stackrechner-Beispiel aufgezogenem Web-Dienst kommen zwei kleine Komplikationen hinzu:

1. Da die einbettende HTML-Seite + das GWT-kompilierte JavaScript i.A. in unterschiedlichen Bereichen (wie im Beispiel) abgelegt sind, sind auch i.A. zwei web.xml-Dateien konsistent zu ergänzen.

Passend zum Beispiel ergänze man in

```
C:\Programme\Apache\Tomcat 5.5\conf\web.xml
```

folgendes

```
<!-- Sec con für AJAX 15.02.2007 -->
    <security-constraint>
      <web-resource-collection>
        <web-resource-name> BuB1502 </web-resource-name>
        <url-pattern>/java/gwt/*</url-pattern>
      </web-resource-collection>
      <user-data-constraint>
        <transport-guarantee>CONFIDENTIAL</transport-guarantee>
      </user-data-constraint>
    </security-constraint>
```

und in

```
D:\www\webAppsDev\WEB-INF\web.xml
```

folgendes

```
<!-- Sec con für AJAX 15.02.2007 -->
    <security-constraint>
      <web-resource-collection>
        <web-resource-name>AJAX GWT BuB</web-resource-name>
        <url-pattern>/servlet/RemoteCalc</url-pattern >
      </web-resource-collection>
      <user-data-constraint>
        <transport-guarantee>CONFIDENTIAL</transport-guarantee>
      </user-data-constraint>
      <auth-constraint>
         <role-name>FB3Doz</role-name>
         <role-name>Kernteam</role-name>
      </auth-constraint>
    </security-constraint>
```

Die andere Komplikation bzw. Bedingung zeigen diese beiden Einträge beispielhaft:

2. Die `user-data-constraint` müssen für die HTML-Seite + JavaScript einerseits und das Servlet andererseits gleich gesetzt sein.

Ohne Einhaltung dieser Bedingung wird i.A. das Servlet wegen der Verletzung der „gleicher-Host-gleicher-Port“-Bedingung nicht geliefert.

Wie das Beispiel auch zeigt, dürfen Bild 7: Authentifizierung nach Knopfdruck die auth`-constraint` für beide Bereiche durchaus unterschiedlich sein. In der gezeigten Beispieleinstellung wird SSL für Beides gefordert, die Authentifizierung als Mitglied der (hier, vgl. [5], Windows-Server2003-Domain-) Gruppen *FB3Doz* oder *Kernteam* hingegen nur für das Servlet.

Die Wirkung ist, dass man die einbettende Seite und das Startbild mit allen GWT-Widgets zwar über SSL und ggf. erst nach Akzeptieren des Server-Zertifikat zu sehen bekommt, das aber dann einfach „so“und ohne Authentifizierung.

Eine Authentifizierung wird mit einer solchen Einstellung erstmalig nach der ersten Bedienung, sprich hier dem ersten Knopfdruck auf dem Rechner, verlangt. Ein solches Verfahren kann bei B&B-Anwendungen gelegentlich gewünscht sein.

6. Résumée

Das Google Web-Toolkit wurde installiert und einfache Web-Seiten und -Dienste mit AJAX-features erstellt. Diese wurden in einem vorhandenen ([5]) Tomcat-Web-Server bzw. J2EE-Container gestellt. Dabei wurde dessen Rechte- und Protokollverwaltung (<user-data-constraint>, <auth-constraint>) auch für GWT-AJAX genutzt.

Dass letzteres geht und auch mit GWT-AJAX beherrscht wird, ist schließlich eine Grundvorraussetzung für einen ernsthaft eingesetzten Web-Dienst, beispielsweise für das Bedienen und Beobachten (B&B) von Prozessen.

Das ist der Einstieg in AJAX mit Tomcat (J2EE) unter Verwendung von GWT (und Eclipse) — nicht mehr, aber auch nicht weniger.

7. Literatur

[1] Ed Ort and Mark Basler, AJAX Design Strategies, SUN 2006
http://java.sun.com/developer/technicalArticles/J2EE/AJAX/.../design-strategies.pdf

[2] Brett McLaughlin, Mastering Ajax, Part 1..4, IBM, 2005
http://www-128.ibm.com/developerworks/web/library/wa-ajaxintro.html

[3] Albrecht Weinert, Zur Installation des JDK (Java Development Kit)
http://a-weinert.de/weinert/pub/java-install.txt

[4] Albrecht Weinert, Java — Tipps und Tricks
http://a-weinert.de/weinert/pub/java-tips.txt

<5> Albrecht Weinert, Tipps zu Tomcat (für Windows)
http://a-weinert.de/weinert/pub/tomcat-tips.pdf

[6] Albrecht Weinert, Tipps zu CVS für Windows — cvsNT
http://a-weinert.de/weinert/pub/cvsnt-tipp.txt

[7] Google, Web-Toolkit, online-Dokumentation (nicht am Stück verfügbar)
http://code.google.com/webtoolkit/documentation/.

[8] Albrecht Weinert, Tipps zu JMX mit SSL
http://a-weinert.de/weinert/pub/jmx-ssl-tips.pdf

[9] Albrecht Weinert, Windows 2003 Domain
Migration von NT4 mit Fremd-DNS
http://www.a-weinert.de/weinert/pub/w2k3domain.pdf

[10] Albrecht Weinert, Windows Server 2003 — Domain FB3-MEVA
Schulungsräume und Infrastruktur — Renovierung 2007
http://www.a-weinert.de/weinert/pub/fb3-meva-domain2007.pdf

[11] Albrecht Weinert, Tipps zu Tomcat (für Windows)
([13] stattdessen für Tomcat >= 6) http://a-weinert.de/weinert/pub/tomcat-tips.pdf

[12] Albrecht Weinert, Windows Server 2003 — Domain FB3-MEVA
Workstations und Server — Renovierung 2007
http://www.a-weinert.de/weinert/pub/fb3-meva-workst2007.pdf

[13] Albrecht Weinert, Tomcat — mit Windows und Active Directory
(ersetzt [11] als Nachfolger) http://www.a-weinert.de/weinert/pub/tomcat-win-ad.pdf

Hinweis: Aus Dateien „.../docu/*.txt“ könnten inwischen teilweise „.../weinert/pub/*.pdf “ geworden sein.